

AMENDMENT OF CLAIMS

(Claim 1, amended)

1. A parallel processing method for performing processing tasks in parallel on a plurality of processors comprising:

(a) identifying at least one area of a large processing task directed to a plurality of computational processes that can be grouped together as a task space not dependent on passing of control of processing from an external process in order to complete processing of the computational processes of the task space;

(b) breaking down the a-processing task space into a plurality of self-contained task objects each of which can be executed in one computational step without requiring passing of control to or from another object, wherein each task object is defined with a computational task step and at least one "data-waiting" slot for receipt of data requested from another task object to which the processing aforesaid task object passes a message for the requested data, and wherein once all the "data-waiting" slots of a task object are filled by the respective return message(s), the task object can perform its defined computational task step without waiting for any other input;

(b c) scheduling the defined task objects of said identified task space to be processed by distributing them across any of the plurality of processors, by:

- (i) placing a task object with an unfilled "data-waiting" slot in a "waiting" state in which it is not assigned to any processor;
- (ii) changing the status of a task object to an "active" state when all of its defined "data-waiting" slots have been filled, wherein it is assigned to a next available processor in an "unoccupied" state, then placing that processor's status in an "occupied" state; and
- (iii) changing the status of the task object to a "dead" state when the computational task step to be performed for the task object by the assigned processor has been completed, and then changing the processor's status to an "unoccupied" state to be assigned to a next "active" task object.

(Claim 2, amended)

2. A parallel processing method according to Claim 1, wherein a master task grouping is defined by a plurality of task spaces each of which contains multiple task objects and does not require passing of control from an external source in order to complete computation for the respective task space.

(Claim 3, original)

3. A parallel processing method according to Claim 2, wherein all task objects of the task spaces which are in an "active" state are placed in a processing queue and each is assigned in turn to a next available "unoccupied" processor.

(Claim 4, original)

4. A parallel processing method according to Claim 3, wherein a master engine for the master task grouping maintains threads which track the processing of task objects in each of the task spaces.

(Claim 5, original)

5. A parallel processing method according to Claim 4, wherein the master engine for the master task grouping maintains an internal space address assigned to each respective task object.

(Claim 6, original)

6. A parallel processing method according to Claim 5, wherein a task object in one master task grouping can exchange data with a task object in another master task grouping by providing its internal space address indexed to its master task grouping.

(Claim 7, amended)

7. A parallel processing method according to Claim 1, wherein ~~the processing~~ an identified task in computer graphics rendering includes shading an image frame of a scene ~~in~~ in computer graphics rendering.

(Claim 8, original)

8. A parallel processing method according to Claim 7, wherein the shading task

includes a master task grouping of shading task spaces each of which performs shading of a pixel in the image frame.

(Claim 9, original)

9. A parallel processing method according to Claim 8, wherein each shading task space includes a plurality of "pixel shading" task objects for performing shading of the pixel based upon ray shooting from light sources in the scene, and a "compositing" task object for compositing the shading results for the pixel.

(Claim 10, original)

10. A parallel processing method according to Claim 9, wherein each shading task object has at least one "data-waiting" slot for return of data characterizing light emitted from a respective light source in the scene.

(Claim 11, original)

11. A parallel processing method according to Claim 9, wherein the rendering task includes a function for receiving scene data for a "world map" of the scene, a function for defining the scene objects in each frame of the scene, a function for defining the pixels of an object in the scene intersected by an eye ray of a viewer of the scene, and a function for tiling together the shading results returned by each of the master shading task groupings for respective objects in the image frame.

(Claim 12, amended)

12. A software programming method for performing processing tasks in parallel on a plurality of processors comprising:

(a) identifying at least one area of a large processing task directed to a plurality of computational processes that can be grouped together as a task space not dependent on passing of control of processing from an external process in order to complete processing of the computational processes of the task space;

(b) breaking down the a processing task space into a plurality of self-contained task objects each of which can be executed in one computational step without requiring passing of control to or from another object, wherein each task object is defined with a computational task step and at

least one "data-waiting" slot for receipt of data requested from another task object to which the ~~processing aforesaid task object~~ passes a message for the requested data, and wherein once all the "data-waiting" slots of a task object are filled by the respective return message(s), the task object can perform its defined computational ~~task step~~ without waiting for any other input;

(b) defining the task objects to be processed by ~~distributing them across any of~~ the plurality of processors, by:

- (i) placing a task object with an unfilled "data-waiting" slot in a "waiting" state in which it is not assigned to any processor;
- (ii) changing the status of a task object to an "active" state when all of its defined "data-waiting" slots have been filled, wherein it is assigned to a next available processor in an "unoccupied" state, then placing that processor's status in an "occupied" state; and
- (iii) changing the status of the task object to a "dead" state when the computational ~~task step~~ to be performed for the task object by the assigned processor has been completed, and then changing the processor's status to an "unoccupied" state to be assigned to a next "active" task object.

(Claim 13, amended)

13. A software programming method according to Claim 12, wherein a master task grouping is defined by a plurality of task spaces each of which contains multiple task objects and does not require ~~message passing of control~~ from an external source in order to complete computation for the respective task space.

(Claim 14, original)

14. A software programming method according to Claim 13, wherein all task objects of the task spaces which are in an "active" state are placed in a processing queue and each is assigned in turn to a next available "unoccupied" processor.

(Claim 15, original)

15. A software programming method according to Claim 14, wherein a master engine for the master task grouping maintains threads which track the processing of task objects in each of

the task spaces.

(Claim 16, original)

16. A software programming method according to Claim 15, wherein the master engine for the master task grouping maintains an internal space address assigned to each respective task object.

(Claim 17, original)

17. A software programming method according to Claim 16, wherein a task object in one master task grouping can exchange data with a task object in another master task grouping by providing its internal space address indexed to its master task grouping.

(Claim 18, original)

18. A software programming method according to Claim 12, further comprising storing templates for different types of task engines, spaces, and objects in a library and utilizing the templates to generate software programming for a desired processing task.

(Claim 19, amended)

19. A software programming method according to Claim 12, wherein ~~the processing~~ ~~an identified task in computer graphics rendering~~ includes shading an image frame of a scene ~~in computer graphics rendering~~.

(Claim 20, original)

20. A software programming method according to Claim 19, wherein the shading task includes a master task grouping of shading task spaces each of which performs shading of a pixel in the image frame.